Nondimensional Simplification of Tensor Polynomials with Indices

A. Balfagón[*]and X. Jaén.[†]

Abstract

We are presenting an algorithm capable of simplifying tensor polynomials with indices when the building tensors have index symmetry properties. These properties include simple symmetry, cyclicity and those due to the presence of covariant derivatives. The algorithm is part of a *Mathematica* package called *Tools of Tensor Calculus (TTC)*[web address: http://baldufa.upc.es/ttc]

1 Introduction

The two main languages commonly used in writing tensor calculus expressions are the intrinsic notation and the index notation. The intrinsic notation seems to be the preferred language for making computer algorithms work with symbolic tensor expressions. The index notation, on the other hand, is extremely powerful in expressing and manipulating tensors. There are some simple expressions (in index notation) which are difficult to express in intrinsic notation for the purpose of introducing new operators with new properties. This is the case e.g. expressing the simple operation of raising an index (1)

$$T_{i\ k}^{\ j} = g^{jm} T_{imk} \tag{1}$$

Recently some computer tools capable of working with index notation at a symbolic level have appeared [1, 2, 3].

In the present study we are facing the problem of tensor polynomial simplification using index notation. In a recent article [3] we dealt with this problem, restricted to the case of simple index symmetry properties of the building tensors like

$$T_{i_{\sigma(1)}\ldots i_{\sigma(n)}} + a\; T_{i_1\ldots i_n} = 0 \tag{2}$$

a being a scalar and σ a permutation of $\{1, ..., n\}$. (2) includes symmetry and antisymmetry of any sequence of indices, and pairsymmetry. We will refer to

[*] Institut Químic de Sarrià, Laboratori de Física Matemàtica, Societat Catalana de Física (I.E.C.)Universitat Ramón Llull e-mail: abalf@iqs.url.es

[†] Universitat Politècnica de Catalunya, Laboratori de Física Matemàtica, Societat Catalana de Física (I.E.C.) e-mail: jaen@baldufa.upc.es

these kinds of properties as *monoterm.* The algorithm proposed in [3] does not use any kind of library, either temporal or permanent, and for that reason it does not require a lot of computer memory. Nevertheless, beside the monoterm restriction the algorithm presented did not draw benefit from all the calculus done. It was reasonably fast on one monomial but not on a hundred, which is a problem to be solved when using other properties.

V.A. Ilyn and A.P.Kryukov [2] have proposed a method capable of working with index symmetry properties, which include cyclic type properties like

$$T_{ijk} + T_{kij} + T_{jki} = 0 \tag{3}$$

(3) was included by the authors in a set called *multiterm* like properties. Although the problem was theoretically satisfactorily solved, the computer time and the amount of computer memory needed to make elementary simplifications was large enough to consider that this study may have some practical and fundamental limitations (since an expression with 11 indices needs about 2500Mb).

One of the problems encountered by V.A. Ilyn and A.P.Kryukov is related to the dummy indices. They consider the possibility of renaming dummy indices as new relations to add to the set of relations generated using the true index properties of building tensors. So in the monomial $S_{jk}T^{kji}$ as well as the properties due to the fact that S is a symmetric tensor they add the ones deduced from the fact that the indices j, k are dummy, i.e.,

$$S_{jk}T^{kji} - S_{kj}T^{jki} = 0 \tag{4}$$

The number of this kind of relations increases factorially with the number of dummy indices, so the algorithm needs to work very hard for monomials of interest. Other authors [4, 5] have studied simplification algorithms but these need to be implemented. R.Portugal [4] handles cyclic like properties similar to [3]. We think that his algorithm can work very well for monoterm properties (like [3] does) but not for multiterm ones. The trouble is that although apparently he can decide if a multiterm rule takes advantage using them increasing the lexicographic order of the working monomial, it can happen that this decision must be delayed after analyzing the effect of some chain of multiterm rules applied to the monomial. In general *the whole set of monoterm plus multiterm monomial equations must be considered.*

The algorithm that we present here, written in *Mathematica* [6] language, is a part of the package *Tools of Tensor Calculus (TTC)* [7, 8, 3, 9](from now on we will call both, the algorithm and the package, *TTC*). *TTC* can work with monomials whose building tensors have both monoterm and multiterm properties. In addition to dealing with dummy indices it also handles symmetry (including the partial derivatives, although it is not recommended because of their non-tensor character and the increase of the corresponding computer time. In this paper we will leave aside partial derivatives) and antisymmetry of any sequence of indices, pairsymmetry, cyclic type properties and the Ricci commuting relations due to the presence of two or more covariant derivatives.

Taking the Riemann tensor as a paradigm *TTC* can handle monomials that have this tensor as a factor, among others, with the known properties:

$$
\begin{aligned}
R_{ijkl} &= -R_{ijlk} \qquad (5)\\
R_{ijkl} &= -R_{jikl} \qquad (6)\\
R_{ijkl} &= R_{klij} \qquad (7)\\
R_{ijkl} + R_{iljk} + R_{iklj} &= 0 \qquad (8)\\
R_{ijkl;m} + R_{ijmk;l} + R_{ijlm;k} &= 0 \qquad (9)\\
R_{ijkl\underbrace{;m....;n}_{k}} + a\, R_{ijkl;\underbrace{n....;m}_{k}} + (\text{Riemann terms})_{\underbrace{;...}_{k-2}} &= 0 \qquad (10)
\end{aligned}
$$

where (10) refers to Ricci commuting relations. We have left the inclusion of the properties derived from the dimension of the base space, for the Riemann tensor and for the generic case, for a future work [10].

We have applied the algorithm presented in a real calculus (superenergy tensor problems) finding new relevant results [11].

The plan of the present paper is as follows: in section 2 we present the problem and explain our solution from the theoretical point of view. In section 3 we describe how the algorithm deals with Ricci commuting relations. Section 4 is devoted to describe the main function of the algorithm implementation. In the Appendix A we show some examples of how *TTC* works in practice.

2 The simplification problem

The problem can be stated as follows:

Given a tensor monomial written in index notation together with all index properties of the building tensors, to find if it is zero and, if not, a unique canonical equivalent polynomial.

The canonical equivalent polynomial is derived from the complete set of monomial relations generated from the exhaustive application of all index properties of the tensor factors over the original monomial. The solving procedure is related to some ordering criterion of the set of monomials. The concrete ordering criterion is not important, we will assume that we have a *unique* ordering criterion.

Free and dummy indices

Let $M(I_F^\beta, J_D^\alpha)$ be a given tensor monomial, with F free indices $I_F^\beta = \{j_{\beta(1)}, ..., j_{\beta(F)}\}$ and D dummy indices $J_D^\alpha = \{j_{\alpha(1)}, ..., j_{\alpha(D)}\}$, α and β being any permutation of $\{1, ..., D\}$, $\{1, ..., F\}$ respectively. Example:

$$M(\{i,k,l\},\{j,m,p\}) = S_{ijk} T^{j\ \ m}_{\ p} T^{p}_{\ lm}$$

Due to the meaning of the dummy indices the following identities hold

$$M(I_F^\beta, J_D^\alpha) = M(I_F^\beta, J_D) \quad \forall\alpha \tag{11}$$

J_D being the canonical ordering of dummy indices in the monomial. This means that there will be many equivalent expressions for a given monomial. This is the first problem to be solved.

In [3] we looked at a method in which a monomial could be rewritten in a canonical form with respect to the dummy indices independently of the number of dummy indices present in the monomial, without using much computer memory.

Essentially the method consists of using an ordering criterion that can be applied to a monomial without prior consideration of all the monomials to be ordered. The method presented here is an improved version of [3], in order to optimize the amount of storing data and computer time, following the criteria:

1)Free indices are renamed: `IndexF[1], IndexF[2],...` all of them in upper position (contravariant)

2) Dummy indices are renamed `IndexD[1], IndexD[2],...` all of them in upper position (contravariant)

3) We find the permutations of free, β, and dummy, α, indices in order to find the most ordered monomial expression. Note that at this stage the result is independent of the original order of free indices. We call this *globalcodification*:

$$M(I_F^\beta, J_D^\alpha) \overset{globalcodification}{\Longrightarrow} M(I_F, J_D) = M(I_F) \tag{12}$$

being I_F the canonical ordering of free indices in the monomial.

$M(I_F)$ is the skeleton of the monomial. $M(I_F)$ carries all the necessary and sufficient information on the monomial independent of the name of dummy and free indices. Different monomial skeletons will be denoted by $M_1(I_F)M_2(I_F)...$

Finally we take the permutation β performed on free indices over the *globalcodification* of the monomial. The final result is called *codification* of the original monomial and the process will we called (0)-rules.

$$M(I_F^\beta, J_D^\alpha) \overset{(0)-\text{rules}}{\Longrightarrow} M(I_F^\beta, J_D) = M^{(0)}(I_F^\beta) \tag{13}$$

Note that the canonical sets I_F and J_D are defined through (12) since they depend on the skeleton of the monomial. Also note that any permutation of free indices of a *globalcodification* corresponds to a *codification* version of the corresponding permutation of free indices in the original monomial, so when a monomial has many free indices this procedure optimizes computer time and memory.

On the other hand, as we will see later, all properties found during a calculation are stored in *codification* version but all monomials with the same *globalcodification* version can use, due to what we have stated on the precedent paragraph, the corresponding β-like permutation version of the stored properties, so the work of finding properties is essentially independent of the order of free indices.

In the following sections we will handle monomials in the form $M^{(0)}(I_F^\beta)$. In this way we will not mention the dummy indices problem anymore.

Tensor properties

Of all tensor properties we will emphasize those that will be of importance within the algorithm.

- **Monoterm properties**: the result of their application over a given monomial is also a monomial.
- **Multiterm properties**: the result of their application over a given monomial is a polynomial (more than one term).
- **Homogeneous properties**: they are properties of only one tensor, say $T_{i_1 \ldots i_n}$, of the form

  $$T_{i_1 \ldots i_n} + \sum_{j=1}^{M} a_{\sigma_j} T_{i_{\sigma_j(1)} \ldots i_{\sigma_j(n)}} = 0 \tag{14}$$

  σ_j being permutations of indices and a_{σ_j} scalars, in such a way that (14) is invariant under any of the σ_k^{-1}, that is

  $$\begin{aligned} & T_{i_{\sigma_k^{-1}(1)} \ldots i_{\sigma_k^{-1}(n)}} + \sum_{j=1}^{M} a_{\sigma_j} T_{i_{(\sigma_k^{-1} \circ \sigma_j)(1)} \ldots i_{(\sigma_k^{-1} \circ \sigma_j)(n)}} = \\ & A_k \left(T_{i_1 \ldots i_n} + \sum_{j=1}^{M} a_{\sigma_j} T_{i_{\sigma_j(1)} \ldots i_{\sigma_j(n)}} \right) \quad \forall k = 1, \ldots M \end{aligned} \tag{15}$$

  A_k being some scalars. Example: see (3).
- **Inhomogeneous properties** : These are properties relating one or more tensors whose form is not reducible to (14) fulfilling (15).

The most important example of inhomogeneous properties is the Ricci commuting relation. The simplest case is when we have a vector field v, then

$$v_{i;j;k} - v_{i;k;j} - v_m \, R^m{}_{i\,j\,k} = 0 \tag{16}$$

R being the Riemann tensor

The advantage of handling homogeneous properties is that we can pick up a term and use it as a pattern to be applied when finding rules over a given monomial. This procedure, due to the homogeneous character, will be equivalent to use all terms in substitutions on the monomial. This is not true for inhomogeneous properties and special treatment must be considered.

On the other hand some inhomogeneous properties can be applied directly *iff* the right hand side inherits all the properties carried from the left hand side. There is an important example when defining the Ricci tensor. $R^m_{imj} = R_{ij}$ can be applied directly *iff* Ricci inherit from Riemann all the resulting properties, for example

$$R^m_{i;m} = \frac{1}{2} R^m_{m;i} \tag{17}$$

which is a consequence of applying Ricci definition to the original Bianchi II relation. If we want to use the definition of the Ricci tensor as a direct relation we must include also (17) as a direct relation, as well as the deduced properties belonging to the Ricci tensor itself, $R_{ij} - R_{ji} = 0$. Although such a procedure can be difficult and unnecessary because we can work exclusively with the Riemann tensor, it is very convenient in order to eliminate superfluous indices.

Induced monomial properties

Given a set of monomials $M = \{M_1^{(0)}(I_F^\beta), M_2^{(0)}(I_F^\gamma), ...\}$ and a set of properties $P = \{p_1, ...p_k..\}$ of the building tensors, the set of monomial properties induced by P, $P(M)$, is the set of properties which arise applying P over M and over *all new monomials generated until no new monomials arise.*

The essential idea when simplifying a given monomial $M^{(0)}(I_F^\beta)$ is to take the set of properties of the building tensors, P, and generate the set of monomial properties $P(\{M^{(0)}(I_F^\beta)\})$, solving them with respect to the ordering criterion and finally applying the corresponding rules onto the original monomial. The trouble is that the set $P(\{M^{(0)}(I_F^\beta)\})$ can be too large to be reasonably generated and solved.

It is interesting to try to break the set of properties P into two sets, say P_1 and P_2. Then generate and solve $P_1(\{M^{(0)}(I_F^\beta)\})$ applying the resulting rules over $M^{(0)}(I_F^\beta)$ giving, in general, a polynomial $Pol_{(1)}$ built by the set of monomials $\{..., M_i^{(1)}(I_F^\beta), ...\}$. Then generate and solve $P_2(\{..., M_i^{(1)}(I_F^\beta), ...\})$ to give a new set of rules to be applied onto the polynomial $Pol_{(1)}$ giving the final result.

We will call *separable properties* the set P_1 of P which can be separated from the rest in the sense of the above paragraph. The separability is a possible property of the given set P which must be demonstrated case by case. Properties which affect different tensors or different sets of indices in the same tensor are clearly separable, but this is not only possibilitu. It is easy to assess whether a given monoterm property is separable or not. The most important case is the Riemann tensor

Lemma 1 *From the whole set of properties of the Riemann tensor, $R_{ijkl} + R_{ijlk} = 0$ and $R_{ijkl} + R_{jikl} = 0$ are separable.*

Proof: We use the Riemann properties $P = \{(5), (6), (7), (8), (9), (10)\}$. Let $P_1 = \{p_1, p_2\} = \{(5), (6)\}$ and $P_2 = P - P_1$.

Given any monomial having the Riemann tensor as a factor and using P_1 to improve the ordering, can be written as

$$RF(i, j, k, l) = ...R_{ijkl}...$$

where the indices $ijkl$ can be free or dummy but we can consider that the actual names will correspond to the most orderly form. P_1 are separable *iff* $P_2(\{RF(i, j, k, l)\}) = P_2(\{RF(i, j, l, k)\}) = P_2(\{RF(j, i, k, l)\})$

Let us analyze P_2 case by case. We will use the notation $(\textit{ref.})(\{M\})$ to indicate the set of properties $P_i(M)$ being P_i a property of the set P:

pairsymmetrie (7)

Clearly $\{(5),(6)\}$ is separable from the set $\{(5),(6),(7)\}$.

Bianchi I (8)

$(8)(\{RF(i,j,k,l)\})$ gives

$$RF(i,j,k,l) + RF(i,l,j,k) + RF(i,k,l,j) = 0 \tag{18}$$

$(8)(\{RF(i,j,l,k)\})$ gives $RF(i,j,l,k)+RF(i,k,j,l)+RF(i,l,k,j) = 0$ which after using $\{(5),(6)\}$ coincides with (18)

$(8)(\{RF(j,i,k,l)\})$ gives $RF(j,i,k,l) + RF(j,k,l,i) + RF(j,l,i,k) = 0$ which after using $\{(5),(6)\}$ gives

$$-RF(i,j,k,l) - RF(j,k,i,l) + RF(j,l,i,k) = 0 \tag{19}$$

which does not coincide with (18) so p_2 is not separable from Bianchi I, but since we have also pairsymmetrie we can consider $(8)(\{RF(k,l,i,j)\})$ which gives (19) so we can conclude that $\{(5),(6)\}$ is separable from the set $\{(5),(6),(7),(8)\}$.

Bianchi II (9)

Considering a monomial of the form $RF(i,j,k,l,m) = ...R_{ijkl;m}....$ and finding $(9)(\{RF(j,i,k,l,m)\})$ and $(9)(\{RF(i,j,l,k,m)\})$ the properties generated are the same as the property generated by $(9)(\{RF(i,j,k,l,m)\})$, so $\{(5),(6)\}$ is separable from the set $\{(5),(6),(7),(8),(9)\}$.

Ricci commuting relations (10)

It is easy to observe that the (Riemann terms) of (10) have the same symmetry properties with respect to the indices i,j,k,l as the term $R_{ijkl;m;n}$, specially $\{(5),(6)\}$, so clearly $\{(5),(6)\}$ is separable from the set $\{(5),(6),(7),(8),(9),(10)\}$, and this completes the proof.

TTC uses this separability property when internally defining the Riemann tensor.

In general we assume two sets of properties accordingly to the possibility of separating some from the whole set. We will call them (n)-properties with $n = 1,2$

How to handle (n)-properties

Here we explain how *TTC* finds, solves and applies (1)-properties and (2)-properties. We will supose that we depart from a monomial $M_1^{(0)}(I_F^{\beta_1})$ so let $n = 1$ and go to **step** n.

Step n:

Given a monomial $M_1^{(n-1)}(I_F^{\beta_1})$ the exhaustive application of the (n)-properties generates a set of relations of the type

$$M_i^{(n-1)}(I_F^{\beta_p}) + \sum_{jq} b_{ijpq} M_j^{(n-1)}(I_F^{\beta_q}) = 0 \quad i,j = 1, ... \tag{20}$$

where i, j, p, q enumerate possible M, β, b objects appearing in (20) and $b_{...}$ are scalars.

This set of relations can always be solved in favor of the subset of the most orderly monomials

$$M_i^{(n-1)}(I_F^{\beta_p}) = \sum_{j,q} c_{ipjq} M_j^{(n)}(I_F^{\beta_q}) \tag{21}$$

c_{ipjq} being scalars and $M_j^{(n)}(I_F^{\beta_q})$ the most orderly monomials from the set $M_i^{(n-1)}(I_F^{\beta_r})$ of (20) . The specific way to solve (20) will be explained in section 4.

Finally we apply the obtained (n)-rules over the original monomial

$$M_1^{(n-1)}(I_F^{\beta_1}) \overset{(n)-\text{rules}}{\Longrightarrow} \sum_{j,q} c_{11jq} M_j^{(n)}(I_F^{\beta_q}) \tag{22}$$

(n)-rules are conveniently stored for further use, as we will see later

If $n < 2$ set $n = n + 1$ and go to **step** n

3 Ricci commuting relations

As we have mentioned, Ricci commuting relations are inhomogeneous properties which must be handled with special care. If we want the algorithm to work correctly it is not sufficient to take into account Ricci commuting relation in the form (where we are taking an example of a single vector field v)

$$v_{i;j;k} \to v_{i;k;j} + v_m \, R^m{}_{i\,j\,k} \tag{23}$$

since these rules will not benefit from other rules one might have $v_{i;j...}$ in the case of monomials which do not have covariant derivatives explicitly, as in

$$v_m \, R^m{}_{i\,j\,k} v_l$$

To solve this problem we have built a function, called `AntiRicci` which takes any monomial and gives all the monomials related to it by Ricci commuting relations *via* covariant derivatives. After that we apply Ricci commuting relations over the covariant derivatives.

$$\texttt{AntiRicci}[v_m \, R^m{}_{i\,j\,k}] = \{v_{i;j;k}\} \tag{24}$$

With respect to separability we can write

Lemma 2 *Given a tensor $T_{ijkl...}$ all symmetry properties of indices $(i, j, k, l..)$ are separable from the Ricci commuting relations*

proof: Is trivial due to the fact that Ricci commuting relations are invariant under these symmetries.

This Lemma does not apply for properties involving covariant derivative indices. There is an important example when defining v_i as Killing vector field: $v_{i;j} + v_{j;i} = 0$ is not separable from Ricci commuting relations.

The examples on the Appendix A are specially addressed these questions.

Ricci commuting relations are internally included in the set of (2)-properties. Accordingly, if a tensor has no-separable properties and we want to use Ricci commuting relations properly, the set of properties must be included as (2)-properties

4 Algorithm description

The algorithm for simplification of polynomial tensors in index notation, called `SimplifyAllIndex`, has the following functions:

Functions related to single tensors

- `InputTensor`:
  `InputTensor[`*Tsymbol,basis,ranklist*`]` declares that the symbol *Tsymbol* will be a tensor in the basis *basis* and with rank and type *ranklist*.

  Example

  ```
  InputTensor[T,XX,{1,1,1,1}]
  ```

- `InputSymmetries[`*Tsymbol*`[`*index*`]`,*symmetriesspecifications*`]` declares that the tensor, previously introduced using `InputTensor`, *Tsymbol* has the symmetry properties of the indices, positioned as indicated by the argument *index*, according to the specifications *symmetriesspecifications*.

  Example 1:

  `InputSymmetries[T[i,j,k,l],{i,k,l}[1]]` declares that `T` is totally symmetric with respect to the indices `{i,k,l}` and add this property to the set of (1)-properties

  Example 2:

  here we declare the same properties as the Riemann tensor, see (5,6,7, 8, 9, 10), for the tensor `T` accordingly to the separability theorem given for the Riemann properties.

  ```
  InputSymmetries[T[i,j,k,l],
  {{i,j}}[1],{{k,l}}[1],{{i,j},{k,l}}[2],Cyclic[j,k,l][2]];

  InputSymmetries[T[i,j,k,l,.;m],Cyclic[k,l,m][2]]
  ```

- `BasicRules[`*n*`]`: once the symmetries of indices have been declared the corresponding rules that implement these symmetries are stored in the list `BasicRules[`*n*`]` . $n = 1$ for (1)-rules and $n = 2$ for (2)-rules. The value $n = 0$ is reserved for the case of properties that can be used directly , like $T_i^i = 0$, which are generally introduced explicitly by the user, or some properties *TTC* already has for Riemann tensor.

Functions related to monomials

In the following we will assume that there are no direct properties stored in `BasicRules[0]` as they are inessential.

The main function is `SimplifyAllIndex`:

`Index[`*metricname*,`SimplifyAllIndex[2]][`*polytens*`]` simplify, i.e. canonize, the polynomial *polytens* taking into account all the declared properties of the building tensor, as well as the dummy indices and the properties related to covariant derivatives. `SimplifyAllIndex[2]` is applied over each term of *polytens*. This function has, in turn, the following internal functions (We will assume that the initial monomial is $M(I_F^\gamma, J_D^\alpha))$:

- `SimplifyAllIndex[0]`: first we canonize the monomial with respect to the dummy indices. This step is taken each time a monomial is the result of some kind of index manipulation. So our initial monomial and all other generated monomials will have the form $M_i^{(0)}(I_F^{\beta_p})$
- `LlistaMonomis[`*label,n,counter(n)* `]`: in these lists all the monomials considered before the present case are stored in *globacodification* version together with its β-permutation which relate it to the way they are present in the corresponding rules. $n = 1$ for (1)-rules and $n = 2$ for (2)-rules. *counter(n)* is a counter which says that the monomials therein are the *counter(n)*-th studied monomials.
- `LlistaRules[`*label,n,counter(n)*`]`: in these lists the simplifying rules of the monomial in the corresponding `LlistaMonomis` are stored.

The specific way *TTC* stores (n)-rules is

`LlistaMonomis[`*label,n,counter(n)*`]` = $\left\{..., \left\{M_i^{(n-1)}(I_F), \beta_i\right\}, ...\right\}$

`LlistaRules[`*label,n, counter(n)*`]` = $\left\{..., \left\{M_i^{(n-1)}(I_F^{\beta_p}) \to \sum_{(j,q)} b_{ipjq} M_j^{(n)}(I_F^{\beta_q})\right\}, ...\right\}$

- `SimplifyAllIndex[{`*n*`}]`, with $n = 1, 2$, simplify a monomial already simplified with respect to $n - 1$ properties, say $M^{(n-1)}(I_F^\gamma)$, with respect to (n)-properties. It searches if its *globalcodification* version, $M^{(n-1)}(I_F)$, is in one of the corresponding lists `LlistaMonomis` and if it is the case it takes, if necessary, the corresponding $(\gamma^{-1} \circ \beta_k)$ permutation version of the `LlistaRules` solving with respect to the ordering criterion and applying the new rules to the monomial $M^{(n-1)}(I_F^\gamma)$.

If $M^{(n-1)}(I_F)$ is not found in any lists `LlistaMonomis` it applies the rules stored in `BasicRules[`n`]` exhaustively generating new lists `LlistaMonomis` and `LlistaRules` and applying them as explained above.

- `PFind` is a simple tool that solves any system of linear equations with a fixed ordering criterion.

  Let $\{x_1, x_2, ..., x_M\}$ be the space of ordered variables to be solved. Let

  $$\texttt{eqlist} = \{c^1 + \sum_j a^{1j} x_j = 0, c^2 + \sum_j a^{2j} x_j = 0, ..., c^N + \sum_j a^{Nj} x_j = 0\} \tag{25}$$

  be the list of N equations with M variables ordered first taking into account the variables and then the coefficients. That is to say, if in the n-th equation the most disordered variable is x_m then x_{m+1} cannot be present in the $(n-1)$-th equation.

  `PFind` takes the initial list of equations and starts the following loop:

  Step 1: `PFind` takes the first equation in `eqlist`. It is solved with respect to the most disordered variable. This result is stored in `eqrule[1]`. `eqrule[1]` is applied over `eqlist` and after eliminating zeros this result is assigned to `eqlist`. Go to step 2

  Step $n > 1$: If `eqlist` is empty the loop is finished. On the contrary `PFind` takes the first equation in `eqlist`. It is solved with respect to the most disordered variable. This result is stored in `eqrule[`n`]`. `eqrule[`n`]` is applied over `eqrule[`i`]` $i = 1, ..., n-1$. `eqrule[`n`]` is applied over `eqlist`, the zeros are eliminated and the result assigned to `eqlist`. Go to the step $n+1$.

  `eqlist` will remain empty if the system is compatible. These must be the case for a set of compatible tensor properties.

The computer time and memory of `PFind` have been controlled.

5 Concluding Remarks

We have solved the problem of simplification of tensor polynomials in index notation with respect to a large set of properties for the building tensor satisfactorily. The computer time and memory needed are sufficiently reasonable so we think that the program can be of interest for practical purposes. We have successfully used the algorithms presented [11]

Due to the generic character of the algorithm (the only requirement is that the input polynomial follows the usual mathematical rules) it can be useful in the kind of fields where tensors are a natural language. As is well known tensor calculus is powerful enough to cover large fields of mathematics, physics and engineering.

There is an important set of properties not included in the present algorithm. These are the ones that appear when the dimension of the space is taken into account. The essential reason for not including these relations is that applying properties of the building tensor do not generate this kind of relations. They arise when the whole monomial is taken into account so we need some tool to generate the dimensional properties for a given monomial. We are leaving this new tool for future work [10].

Acknowledgments

X.J. would like to thank the Comisión Asesora de Investigación Científica y Técnica for partial financial support, under Contract No. PB96-0384. The authors would also like to thank the Laboratori de Física Matemàtica at the Societat Catalana de Física for partial financial support.

References

[1] L.Parker and S.M.Christensen, *Mathtensor A System for Doing Tensor Analysis by Computer* , Addison-Wesley Publishing Company (1994)

[2] V.A.Ilyin and A.P.Kryukov *Computers in Physics Communications* **96**, 36.(1996)

[3] A.Balfagón and X.Jaén, *Computers in Physics*, **12**, 3 , 286,1998.

[4] R.Portugal, To be published on Journal of Physics A

[5] S.A. Fulling, R.C. King, B.G. Wybourne and C.J.Cummins , Class.Quantum Grav. **9** 1151-1197 (1992)

[6] S.Wolfram *Mathematica: a System for Doing Mathematics by Computer*, (Addison-Wesley, redwood City, CA)(1991)

[7] P.Castellví , X.Jaén and E.LLanta, Comput. Phys. **8**,3 (1994)

[8] P.Castellví , X.Jaén and E.LLanta, Comput. Phys. **9**, 3 (1995)

[9] *Tools of Tensor Calculus (TTC) Mathematica* package, web adress http://baldufa.upc.es/ttc

[10] A.Balfagón and X.Jaén preprint

[11] A.Balfagón and X.Jaén preprint

[12] C. W. Misner, K. S. Thorne, J. A. Wheeler, *Gravitation*

[13] D.Kramer, H.Stephani, M.MacCallum, E.Herlt. *Exact solutions of Einstein Field Equations* Editor E.Shumtzer (1980)

Appendix A

We present here an example of how the algorithm `SimplifyAllIndex` can be applied. The session is part of a *Mathematica* notebook. The inputs are indicated by `In[]:=`. The outputs have no prefix. The titles and comments are enclosed using the symbols `(*`*comment*`*)`.

Since we use the Riemann tensor as an example we introduce this tensor using the *TTC* function `InputSRiemann`. This is internally equivalent to use `InputTensor` and `InputSymmetries` to declare the Riemann and Ricci tensors and to give the relation between them and between the curvature scalar. All tensor calculus conventions follow ([12]).

In this first short academic example we first introduce coordinate system `cx4` and a metric `gn` together with the Riemann tensor.

We define the tensor V and L with symmetries $L^{ijk} = L^{kji}$, $L^{ijk;m} = L^{jik;m}$ and $L^{ijk;m} + L^{imj;k} + L^{ikm;j} = 0$. Note that L have not separable properties so we will define all properties as a (2)-properties.

Next we declare the indices for outputs and define and simplify a monomial

```
In[]:=<<ttc.m
 ----------------------------------------
 |  Tools  of  Tensor  Calculus 4.1.0   |
 |  A.Balfagon,P.Castellvi and X.Jaen   |
 |     \protect\vrule width0pt\protect\href{http://baldufa.upc.es/ttc}{http://baldufa.upc.es/ttc}         |
 |     e-mail:ttc@baldufa.upc.es        |
 |    version: september, 29,1999       |
 ----------------------------------------
 |     Session started on               |
 |     October,   5 , 1999              |
 |     at 11 h 15 min 48 s              |
 ----------------------------------------

In[]:=(
InputCoordinates[cx4,4];
InputSMetric[gn,cx4,"g",g];
InputSRiemann[gn,cx4,"R",Rie,Ric,R];
InputTensor[{L},cx4,{1,1,1}];
InputTensor[{V},cx4,{1}];
InputSymmetries[L[i,j,k],{i,k}[2]];
InputSymmetries[L[i,j,k,.;m],{i,j}[2],
Cyclic[j,k,m][2]];
InputIndex[{a,b,c,d,e,m,n}]
);

3/2 L[i,j,k] Rie[-i,-j,q,p] V[-k] //Index[gn]

           c d e             a b
        3 L       V     R
 a b               e      c d
0     + ---------------------------
                     2
%//Index[gn,SimplifyAllIndex[2]]

           b       d    c a
        3 L       V   R
 a b         c d
0     - ----------------------- +
                   4

       b     d      c a
```

```
  3 L       V      R
     c   d
  ----------------------- +
             4
      a       d    c b
  3 L       V    R
       c d
  ----------------------- -
             4

       a     d      c b
  3 L       V      R
     c   d
  ----------------------- +
             4

       a b        c d
  3 L       V    R
     c       d
  ----------------------- -
             4

       b a        c d
  3 L       V    R
     c       d
  -----------------------
             4

(* Note that if Ricci=0 then the original monomial
 is zero, which is not obvious from the beginning*)
```

Next we show how the same kind of calculation is able to find a known result: The vanishing of the magnetic part of the Weyl tensor for static space-times, see [13]

```
(* first we introduce the predefined volume form which we name  H*)
In[]:=
InputSAForm[gn,cx4,"H",H];

(*Here we introduce the Killing field Vt, which we define as
a vector antisymmetric between his own index and
the covariant derivative index. *)

In[]:=
(InputTensor[Vt,cx4,{1}];
InputSymmetries[Vt[a,.;b],{{a,b}}[2]]);

(*Next we introduce the static condition. Since this
property involves more than one tensor, it must be introduced
explicitly by the user and taking into account
its differential consequences.
We introduce these conditions in BasicRules[2]
using the specific syntaxis a_:>(a;b) instead of a_:>b !!*)

(BasicRules[2]=Join[BasicRules[2],{

Vt[a_] Vt[b_,.;c_]:>(Vt[a] Vt[b,.;c];-
           Vt[c] Vt[a,.;b]-Vt[b] Vt[c,.;a]),

Vt[a_] Vt[b_,.;c_,.;d_]:>(Vt[a] Vt[b,.;c,.;d];
        -Vt[c] Vt[a,.;b,.;d]-Vt[b] Vt[c,.;a,.;d]-
              Vt[a,.;d] Vt[b,.;c]-Vt[b,.;d] Vt[c,.;a]-Vt[c,.;d] Vt[a,.;b]),


Vt[b_,.;d_] Vt[c_,.;a_]:>(Vt[b,.;d] Vt[c,.;a];
       -Vt[c] Vt[a,.;b,.;d]-Vt[b] Vt[c,.;a,.;d]-
                 Vt[a] Vt[b,.;c,.;d]-Vt[a,.;d] Vt[b,.;c]-Vt[c,.;d] Vt[a,.;b]),
}]);

(* Next we define Weyl Tensor*)
In[]:=
IndexUpdate[Weyl,":=",
0[a,b,c,d]+Rie[a,b,c,d]-1/2 g[a,c] Ric[b,d]+
1/2 g[a,d] Ric[b,c]+1/2 g[b,c] Ric[a,d]-1/2 g[b,d] Ric[a,c]+
R/12(g[a,c] g[b,d]-g[b,c]g[a,d]+g[b,d]g[a,c]-g[a,d]g[b,c])
//Index[gn]]

(*Next we define the magnetic part of Weyl, BW*)
In[]:=
IndexUpdate[BW,":=",
1/2 H[a,r,c,m] Weyl[-c,-m,b,n] Vt[-r] Vt[-n]//Index[gn]];

(*We can see BW expanded. Now we like to use {i,j,k,l...} for outputs*)
In[]:=InputIndex[{i,j,k,l,m,n,o,p}];

In[]:=BW[-a,-b]//Index[gn,SuperIndexExpand]

                      l   k                   l k                 l k m
        R Vt   Vt   H           R Vt   Vt   H           Vt   Vt   H          R
            k    l   i   j          k    l   i     j      k    l   i          m j
0     + --------------------- - --------------------- + ------------------------- -
 i j             12                      12                          4

               l m k                   l   m     k                 l m       k
  Vt   Vt   H         R       Vt   Vt   H         R       Vt   Vt   H         R
    k    l   i         m j      k    l   i   j     m        k    l   i     j   m
  ------------------------- - ------------------------- + ------------------------- +
              4                           4                           4

               l m n         k
  Vt   Vt   H         R
    k    l   i         m n j
  -----------------------------
              2

(*Finally we simplify BW, which give the final result*)

In[]:=%//Index[gn,SimplifyAllIndex[2]]

0

(*Here we can see the internal aspect of some of the stored rules*)

In[]:=LlistaMonomis[gn,cx4,2,2,2]


    IndexD[1]    IndexF[1] .;IndexD[2] .; IndexD[3]
{{Vt           Vt

    IndexD[1] IndexD[2] IndexD[3] IndexF[2]
  H                                           , {2, 1}},

     IndexD[1]    IndexD[2]  IndexD[1] IndexD[3] IndexD[4] IndexF[1]
  {Vt          Vt          H

  IndexD[2] IndexF[2] IndexD[3] IndexD[4]
 R                                          , {1, 2}},.....


In[]:=
LlistaRules[gn,cx4,2,2,2]


    IndexD[1]    IndexF[2]   IndexD[2] IndexD[3] IndexD[4] IndexF[1]
```

```
{Vt           Vt            H

      IndexD[1] IndexD[2] IndexD[3] IndexD[4]
    R                                         -> 0, .....
```